The ABACOC Algorithm: a Novel Approach for Nonparametric Classification of Data Streams

Rocco De Rosa
Dipartimento di Informatica
Università degli Studi di Milano, Italy

Francesco Orabona
Yahoo Labs
New York, NY, USA

Nicolò Cesa-Bianchi
Dipartimento di Informatica
Università degli Studi di Milano, Italy

***Abstract*—Stream mining poses unique challenges to machine learning: predictive models are required to be scalable, incrementally trainable, must remain bounded in size, and be nonparametric in order to achieve high accuracy even in complex and dynamic environments. Moreover, the learning system must be parameterless—traditional tuning methods are problematic in streaming settings—and avoid requiring prior knowledge of the number of distinct class labels occurring in the stream. In this paper, we introduce a new algorithmic approach for nonparametric learning in data streams. Our approach addresses all above mentioned challenges by learning a model that covers the input space using simple local classifiers. The distribution of these classifiers dynamically adapts to the local (unknown) complexity of the classification problem, thus achieving a good balance between model complexity and predictive accuracy. By means of an extensive empirical evaluation against standard nonparametric baselines, we show state-of-the-art results in terms of accuracy versus model size. Our empirical analysis is complemented by a theoretical performance guarantee which does not rely on any stochastic assumption on the source generating the stream.**[1]

I. Introduction

As pointed out in various papers—e.g. [14], [18]—stream mining poses unique challenges to machine learning: examples must be efficiently processed one at a time as they arrive from the stream, and an up-to-date predictive model must be available at all times. Incremental learning systems are well suited to address these requirements: the key difference between a traditional (batch) learning system and an incremental one is that the latter learns by performing small adjustments to the current predictor. Each adjustment uses only the information provided by the current example in the stream, allowing an efficient and timely update of the predictive model. This is unlike batch learning, where training typically involves a costly global optimization process involving multiple passes over the data. Another important feature of stream mining is that the true structure of the problem is progressively revealed as more data are observed. In this context, nonparametric learning methods, such as decision trees (DT) or nearest neighbour (NN), are especially effective, as a nonparametric algorithm is not committed to any specific family of decision surfaces. For this reason, incremental algorithms for DT [6], [4] and NN [24] are extremely popular in stream mining applications. Since in nonparametric methods the model size keeps growing to fit the stream with increasing accuracy, we seek a method able to improve predictions while growing the model as slowly as possible. However, as the model size cannot grow unbounded, we also introduce a variant of our approach that prevents the model size from going beyond a given limit. In the presence of concept drift [22], bounding the model size may actually improve the overall predictive accuracy. A further issue in stream mining concerns the way prediction methods are evaluated—see, e.g., [12] for a discussion. In this paper, we advocate the use of the online error which measures the average of the errors made by the sequence of incrementally learned models, where one first tests the current model on the next example in the stream and then uses the same example to update the model. Also, the online error does not specifically require stochastic assumptions on the way the stream is generated.

In this paper, we propose a novel incremental and nonparametric approach for the classification of data streams. We present four different instances of our approach characterized by an increasing degree of adaptivity to the data. In a nutshell, our algorithms work by incrementally covering the input space with balls of possibly different radii. Each new example that falls outside of the current cover becomes the center of a new ball. Examples are classified according to NN over the ball centers, where each ball predicts according to the majority of the labels of previous examples that fell in that ball. The set of balls is organized in a tree structure [16], so that predictions can be computed in time logarithmic in the number of balls. In order to increase the ability of the model to fit new data, the radii of the balls shrink, thus making room for new balls. The shrinking of the radius may depend on time or, in the more sophisticated variants of our algorithms, on the number of classification mistakes made by each ball classifier. Similarly to DTs, our method locally adapts the complexity of the model by allocating more balls in regions of the input space where the stream is harder to predict. A further improvement concerns the relocation of the ball centers in the input space: as our methods are completely incremental, the positioning of the balls depends on the order of the examples in the stream, which may result in a model using more balls than necessary. In order to mitigate this phenomenon, while avoiding a costly global optimization step to reposition the balls, we also consider a variant in which a K-means step is used to move the center of a ball being updated towards the median of the data points that previously fell in that ball. A further modification which we consider is aimed at keeping the model size bounded even in the presence of an arbitrarily long stream.

II. Related Work

Within the vast area of stream mining [10], we focus our analysis of related work on the subarea that is most relevant

[1] Full version available at arxiv.org/pdf/1508.04912v1.pdf.

to this study: nonparametric methods for stream classification. The most important approaches in this domain are:

Incremental decision and rule tree learning systems, such as Very Fast Decision Tree (VFDT) [6] and Decision Rules (RULES) [11] which use an incremental version of the split function computation —see also [7], [4].
Incremental variants of NN, such as Condensed Nearest Neighbour (CNN) [23] that stores only the misclassified instances, Lazy-Tree (L-Tree) [24] condensing historical stream records into compact exemplars, and IBLStreams [19], an instance-based learning algorithms removing outliers or examples that have become redundant.
Incremental kernel-based algorithms, such as the kernel Perceptron [9] with Gaussian kernels.[2]

Note that our methods do not belong to any of the above three families: they do not perform a recursive partition of the feature space as DTs, they do not allocate (or remove) instances based on the heuristics used by IBLStreams, and they do not use kernels. As we explain next, our most basic algorithm is a variant for classification tasks of the algorithm proposed in [15] for nonparametric regression in a streaming setting. A similar algorithm was previously proposed in [13] and analyzed without resorting to stochastic assumptions on the stream generation. A preliminary instance of our approach, without any theoretical analysis, was developed in [5].

III. Problem Setting

Our analysis applies to streams of data points belonging to an arbitrary metric space and depends on the *metric dimension* of data points in the stream. This notion of dimension extends to general metric spaces the traditional notions of dimension (e.g., Euclidean dimension and manifold dimension) [2]. The metric dimension of a subset S of a metric space $(\mathcal{X}, \rho)$ is d if there exists a constant $C_S > 0$ such that, for all $\epsilon > 0$, S has an ϵ-cover of size at most $C_S \epsilon^{-d}$ (an ϵ-cover is a set of balls of radius ϵ whose union contains S). In practice, the metric dimension of the stream may be much smaller than the dimension of the ambient space $\mathcal{X}$. This is especially relevant in case of nonparametric algorithms, which typically have a bad dependence on the dimensionality of the data.

The learner receives a sequence $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots$ of examples, where each data point $\boldsymbol{x}_t \in \mathcal{X}$ is annotated with a label y_t from a set $\mathcal{Y} = \{1, \dots, K\}$ of possible class labels, which may change over time. The learner's task is to predict each label y_t minimizing the overall number of prediction mistakes over the data stream. We derive theoretical performance guarantees for BASE, the simplest algorithm in our family (Algorithm 2), without making stochastic assumptions on the way the examples in the stream are generated.

We will analyze the performance of BASE using the notion of *regret* [1]. The regret of a randomized algorithm is defined as the difference between the expected number of classification mistakes made by the algorithm over the stream and the expected number of mistakes made by the best element in a fixed class of randomized classifiers. A randomized binary classifier is a mapping $f : \mathcal{X} \to [0, 1]$, where $f(\boldsymbol{x})$ is the probability of predicting label $+1$. We consider the class $\mathcal{F}_L$ of L-Lipschitz predictors $f : \mathcal{X} \to [0, 1]$ w.r.t. the metric ρ of the space. Namely, $\forall x, x' \in \mathcal{X}, \quad |f(\boldsymbol{x}) - f(\boldsymbol{x}')| \leq L\, \rho(\boldsymbol{x}, \boldsymbol{x}')$. Lipschitz functions are a standard reference in the analysis of nonparametric algorithms. The regret of an algorithm generating randomized predictions $\widehat{y}_t$ is defined by (see also [13])

$$R_L(T) = \sum_{t=1}^{T} \mathbb{P}(\widehat{y}_t \neq y_t) - \min_{f \in \mathcal{F}_L} \sum_{t=1}^{T} \mathbb{P}(f(x_t) \neq y_t) \ .$$

IV. Adaptive Ball Covering

The adaptive ball covering at the roots of our method was previously used in a theoretical work [15]. Here, we distillate the main ideas behind that approach in a generic algorithmic approach (the template Algorithm 1) called ABACOC (Adaptive BAll COver for Classification). We then present our methods as specific instances of this generic template.

The BASE Algorithm. Our first instance of ABACOC is BASE (Algorithm 2), a randomized variant for binary classification of the ITBR (Incremental Tree-Based Regressor) algorithm proposed in [15]. BASE shrinks the radius (line 27) of the balls depending on (1) an estimate of the metric dimension of the stream and (2) the number of data points so far observed from the stream. This implies that the radii of all the balls shrink at the same rate. In the prediction phase, the ball nearest to the input example is considered and a randomized binary prediction is made based on the class distribution estimate locally computed in the ball. Laplace estimators (line 5) and randomized predictions (lines 6–8) are new features of BASE that were missing in ITBR.

For the BASE algorithm we can prove the following regret bound against *any* Lipschitz randomized classifier, without *any* assumption on the way the stream is generated. Moreover, similarly to ITBR, the regret upper bound depends on the unknown metric dimension d of the space, automatically estimated by the algorithm.

Theorem 1: Fix a metric ρ and any stream $(\boldsymbol{x}_t, y_t)$ $t = 1, \dots, T$ of binary labeled points $S = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_T\}$ in a metric space $(\mathcal{X}, \rho)$ of diameter 1 and let d be the metric dimension of S. Assume that Algorithm 2 is run with parameter $\widehat{C} \geq C_S$, where C_S is such that $C_S \epsilon^{-d}$ upper bounds the size of any ϵ-cover of S. Then, for any $L > 0$ we have

$$R_L(T) \leq 1.26 \left(2.5 \sqrt{C_S}\, 2^d + 1.5L\right) T^{\frac{1+d}{2+d}}.$$

The proof is in Appendix A. Note that the algorithm does not know L, hence the regret bound above holds for all values of L simultaneously. This theorem tells us that BASE is not an heuristic, but rather a principled approach with a specific performance guarantee. The performance guarantee implies that, on any stream, the expected mistake rate of BASE converges to that of the best L-Lipschitz randomized classifier at rate of order $T^{-1/(2+d)}$. Next, we generalize the BASE algorithm to multiclass classification and make some modifications aimed at improving its empirical performance.

[2] Gaussian kernels are universal [20], meaning that a kernel-based model can approximate any continuous classification function. Hence, algorithms using Gaussian kernels can be viewed as nonparametric learning algorithms.

Algorithm 1 ABACOC TEMPLATE

Input: metric ρ

`InitProcedure()`
for $t = 1, 2, \dots$ **do**
  Get input example $(\boldsymbol{x}_t, y_t)$
  if $y_t \notin \mathcal{Y}$ **then**
    Set $\mathcal{Y} = \mathcal{Y} \cup \{y_t\}$ ▷ add new class on the fly
  end if
  Let $\mathcal{B}(\boldsymbol{x}_s, \varepsilon_s)$ be the ball in $\mathcal{S}$ closest to $\boldsymbol{x}_t$
  `OuputPrediction`$(\mathcal{B}_s)$
  if $\rho(\boldsymbol{x}_s, \boldsymbol{x}_t) \leq \varepsilon_s$ **then**
    $\mathcal{B}$=`UpdateBallInformation`$(\mathcal{B}_s, (\boldsymbol{x}_t, y_t))$
  else
    $\mathcal{B}$=`AddNewBall`$(\mathcal{S}, \boldsymbol{x}_s, (\boldsymbol{x}_t, y_t))$
  end if
  `UpdateEpsilon`$(\mathcal{B})$
end for

Algorithm 2 BASE

Input: $\widehat{C}$ (space diameter)

procedure INITPROCEDURE
  $\mathcal{S} = \emptyset$, $i = 1$, $t_i = 0$, and $d_i = 1$
end procedure
procedure OUPUTPREDICTION$(\mathcal{B}_s)$
  $q_s = \frac{m_s+1}{n_s+2}$ ▷ laplace estimator of counts
  Set $\gamma_s = \frac{1}{2\sqrt{n_s+2}}$
  Set $p_t = \begin{cases} 0 & \text{if } q_s < \frac{1}{2} - \gamma_s \\ 1 & \text{if } q_s > \frac{1}{2} + \gamma_s \\ \frac{1}{2} + (q_s - \frac{1}{2})/(2\gamma_s) & \text{otherwise.} \end{cases}$
  Predict $\widehat{y}_t = 1$ with probability p_t and 0 otherwise.
end procedure
procedure UPDATEBALLINFORMATION$(\mathcal{B}_s, (\boldsymbol{x}_t, y_t))$
  $m_s = m_s + y_t$ ▷ number of $y_t = 1$ in the ball
  $n_s = n_s + 1$ ▷ total number of points in the ball
end procedure
procedure ADDNEWBALL$(\mathcal{S}, \boldsymbol{x}_s, (\boldsymbol{x}_t, y_t))$
  if $|\mathcal{S}| + 1 > \widehat{C} 2^{d_i} \varepsilon_t^{-d_i}$ **then** ▷ dimension check
    $d_{i+1} = \left\lceil \log\left(\frac{|\mathcal{S}|+1}{\widehat{C}}\right) / \log(2/\varepsilon_t) \right\rceil$ ▷ Phase $i+1$
    $\mathcal{S} = \emptyset$, $i = i + 1$, $t_i = 0$
  end if
  $\mathcal{S} = \mathcal{S} \cup \{\boldsymbol{x}_t\}$
  $m_t = y_t$ ▷ number of $y_t = 1$ in the ball
  $n_t = 1$ ▷ first point in the ball
  $t_i = t_i + 1$ ▷ counts the time steps within phase i
end procedure
procedure UPDATEEPSILON
  $\varepsilon_t = t_i^{-1/(2+d_i)}$ ▷ radius dependent on time
end procedure

The BASE algorithm with ball adjustment. A natural way of generalizing the BASE algorithm to the multiclass case is by estimating the class probabilities in each ball. Note that this approach is naturally incremental w.r.t. the number of classes. The BASE algorithm greedily covers the input space, using balls always centered on input points. However, constraining the centers on data points is an intuitively sub-optimal strategy. We introduce the BASE-ADJ variant which makes a partial optimization of the ball centers. More precisely, BASE-ADJ, moves the center of each ball towards the average of the correct classified data points falling into it —the update procedure is described in line 22 of Algorithm 3.[3] In this way, the center of the ball tends to move towards the centroid of a cluster of points of a certain class. We expect this variant to generate less balls and also to have a better empirical performance. We drop from BASE-ADJ the Laplace correction of class estimates and the randomization in the computation of the predicted label. Hence, BASE-ADJ always predicts the class with the largest class probability estimate within the ball closest to the current data point.

Algorithm 3 AUTO and AUTO-ADJ

Input: $\widehat{d}$

procedure INITPROCEDURE
  // wait until at least two different labels fed
  if $\mathcal{S} \equiv \emptyset$ **then**
    $\mathcal{S} = \{\boldsymbol{x}_1\}$
  else if $y_t \neq y_1$ **then** ▷ $|\mathcal{S}| = 1$
    $\mathcal{S} = \mathcal{S} \cup \{\boldsymbol{x}_t\}, \varepsilon_1 = \varepsilon_t = \rho(\boldsymbol{x}_1, \boldsymbol{x}_t)$
  else
    continue
  end if
end procedure
procedure OUPUTPREDICTION$(\mathcal{B}_s)$
  $n_s = n_s(1) + \cdots + n_s(K)$ ▷ total class counts
  $p_s(k) = \frac{n_s(k)}{n_s} \quad k = 1, \dots, K$
  Predict $\widehat{y}_t = \operatorname{argmax}_{k \in \mathcal{Y}} p_s(k)$
end procedure
procedure UPDATEBALLINFORMATION$(\mathcal{B}_s, (\boldsymbol{x}_t, y_t))$
  // shrink radius on errors
  if $y_t \neq \widehat{y}_t$ **then**
    Set $m_s = m_s + 1$ ▷ update mistakes count
  else if AUTO-ADJ method **then**
    // update ball centre if correct prediction
    $\Delta = \boldsymbol{x}_t - \boldsymbol{x}_s; u_s = u_s + 1; \boldsymbol{x}_s = \boldsymbol{x}_s + \Delta / u_s$
  end if
  Updates label counts $n_s(1), \dots, n_s(K)$ in the ball $\mathcal{B}_s$ using y_t
end procedure
procedure ADDNEWBALL$(\mathcal{S}, \boldsymbol{x}_s, (\boldsymbol{x}_t, y_t))$
  $\mathcal{S} = \mathcal{S} \cup \{\boldsymbol{x}_t\}$, $R_t = \rho(\boldsymbol{x}_t, \boldsymbol{x}_s)$
  $m_t = 0$ ▷ ball mistakes count
  $u_t = 1$ ▷ center updates count (for AUTO-ADJ)
  Init. label counts $n_s(1), \dots, n_s(K)$ in $\mathcal{B}_t$ using y_t
end procedure
procedure UPDATEEPSILON$(\mathcal{B}_s)$
  $\varepsilon_s = R_s \, m_s^{-1/(2+\widehat{d})}$ ▷ radius dependent on mistakes
end procedure

The AUTO algorithm: automatic radius. One of the biggest issues with BASE (and ITBR) is the use of a common radius for all the balls. In fact, we have that the radii ε_s shrink uniformly with time t at rate $t^{-1/(d_i+2)}$, where d_i is the estimated metric dimension. However, we would like the algorithm to use smaller balls in regions of the input space where labels are more irregularly distributed and bigger balls in easy regions, where labels tend to be the same. In

[3] The other procedures are essentially the same as those in BASE.

Data	Cls	Dim	Examples	Drift	Source
sensor	54	5	2,219,803	no	SDMR
kddcup99	23	41	494,021	no	SDMR
powersupply	24	2	29,928	yes	SDMR
hyperPlane	5	10	100,000	yes	SDMR
sea	2	3	60,000	yes	DF
poker	10	10	25,010	no	MOA
covtype	7	54	581,012	yes	MOA
airlines	2	608	539,383	yes	MOA
electricity	2	8	45,312	yes	MOA
connect-4	3	126	67,557	no	LIBSVM
acoustic	3	50	78,823	no	LIBSVM

TABLE I. DATASETS USED FOR BENCHMARKING.

order to overcome this issue, in this section we introduce two other instances of ABACOC: AUTO and AUTO-ADJ. In these variants we let the radius of each ball shrink at a rate depending on the number of mistakes made by each local ball classifier, lines 19 and 33 in Algorithm 3. Moreover, in order to get rid of the parameter $\widehat{C}$, we initialize the radius of each ball to the distance to its closest ball, line 27 in Algorithm 3. In other words, everytime a new ball is added its radius is set equal to the distance to the nearest already-existing ball. AUTO-ADJ differs from AUTO because it implements the same strategy, introduced in BASE-ADJ, for updating the position of the centers. Note that this strategy, coupled with the shrinkage depending on the number of mistakes, makes a ball stationary once it is covering a region of the space that contains data points always annotated with the same label. Using balls of different radii makes it impossible to work with the automatic estimate of the metric dimension used in BASE, BASE-ADJ and ITBR. For this reason, we further simplify the algorithms by resorting to a fixed estimate $\widehat{d}$ of the intrinsic dimension d as an input parameter.

V. EXPERIMENTS

Generally, in real applications for high-speed data streams, when the system cannot afford to revise the current model after each observation of a data point, stream sub-sampling is used to keep the model size and the prediction efficiency under control. In order to emphasize the distinctive features of our approaches (i.e., good trade-off between accuracy and model size), we tested the online (prequential) performance using sub-sampling. In this setting, the algorithms have access to each true class label only with a certain probability which we call `rate` in all the experiments. By varying this probability, we can explore different model sizes for each baseline algorithm and compare the resulting performances.

Baseline and datasets. We considered 11 popular datasets for stream mining listed in Table I. As indicated in the table, datasets are from the Stream Data Mining repository (SDMR) [25], the Data Sets with Concept Drift repository (DF) [21], the Massive Online Analysis (MOA) collection[4], and the LIBSVM classification repository[5]. In all experiments, we measured the *online accuracy* (prequential error in [12] or "Interleaved Test-Then-Train" validation in MOA[6]). This is the average performance when each new example in the stream is predicted using the classifier trained only over the past examples in the stream. In a pre-processing phase, the categorical attributes were binarized. BASE and BASE-ADJ received normalized input instances (Euclidean norm) allowing the input parameter $\widehat{C}$ (space diameter) to be set to 1. We compared our ABACOC methods BASE[7] and BASE-ADJ (Algorithm 2), AUTO and AUTO-ADJ (Algorithm 3) against some of the most popular incremental nonparametric baselines (see Section II) in the stream mining literature: K-NN with parameter $K = 3$ (NN3) (see next paragraph for a justification of this choice), Condensed Nearest Neighbor [23] (CNN), a streaming version of NN which only stores mistaken points, the multiclass Perceptron with Gaussian kernel [3] (K-PERC), a DT for streaming data [6] (VDFT), and a recent algorithm for learning decision rules on streaming data [11] (RULES). For VDFT and RULES we used the implementation available in MOA, while K-PERC was run using the code in DOGMA [17]. The ABACOC algorithms were implemented in MATLAB.[8] We did not consider the L-Tree [24] and IBLStreams [19] methods described in Section II as L-Tree is an efficient approximation of NN (outperformed by NN, see [24]) and IBLStreams never performs better than RULES (both implemented in MOA) on our datasets. Where necessary, the parameters of the competitor methods were individually tuned on each dataset using an algorithm-specific grid of values in order to obtain the best online performance. Hence, the results of the competitors are not worse than the ones obtainable with a tuning of the parameters using standard cross-validation methods. For our methods, we used the Euclidean distance as metric ρ. Based on preliminary experiments, we noticed that the parameter $\widehat{d}$ does not affect significantly the performance in AUTO and AUTO-ADJ, so we set it to 2. With $\widehat{d}$ fixed to this value, our methods are essentially parameterless.

Results. We now turn to describing the sub-sampling experiments. In a streaming setting, the model size and thus the computational efficiency of the prediction system is a key feature. The goal of the experiments is to show the trade-off between online performance and model size for each algorithm. The model size is measured by: the number of balls used to cover the feature space (ABACOC), the number of stored instances (K-PERC, NN, CNN), the number of leaves (VFDT) or rules (RULES) used to partition the feature space. We ran all the methods using values `rate` $= \{1\%, 3\%, 5\%, 10\%\}$ and the same random seeds for all algorithms.[9] In Figure 1, we plot the normalized online performance against model size, averaged over the datasets. The model size is relative to the stream length, whereas the online performance is measured relative to the top-performing method on each dataset without restriction on model size. As we can see from the plot, NN3 saturates the model size and achieves a slightly better overall performance on the larger model sizes. However, it suffers at low budget values and small model sizes. CNN works better than K-PERC and DTs. VFDT and RULES use very little memory but have a worse performance than the other methods. BASE-ADJ improves on the performance of BASE. AUTO attains a better performance than BASE-ADJ and AUTO-ADJ achieves the overall best trade-off between accuracy and model size. In fact, as we can see in Figure 1, the AUTO-ADJ curve dominates the

[4] moa.cms.waikato.ac.nz/datasets/

[5] www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[6] moa.cms.waikato.ac.nz/

[7] We used the multiclass version as for BASE-ADJ.

[8] Code available at http://mloss.org/software/view/560/.

[9] We remark that the `rate` is only an upper bound on the model size. In fact, the methods can select a smaller fraction of data to represent the model.

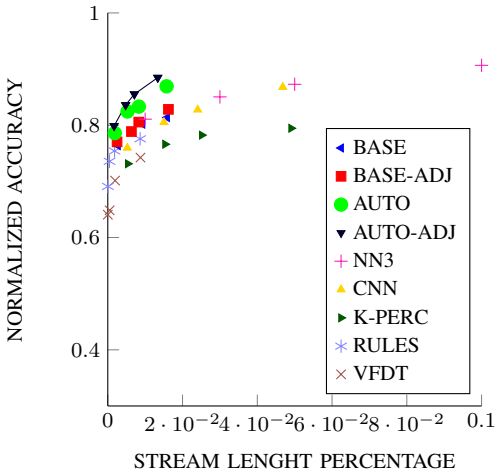

Fig. 1. Online performance against model size averaged over the datasets. The model size is relative to the stream length, whereas the online performance is measured relative to the top-performing method on each dataset without restriction on model size.

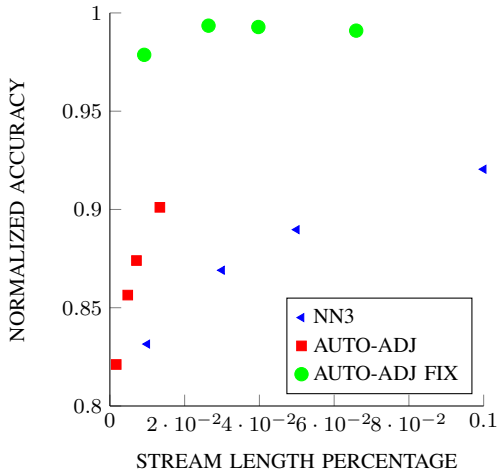

Fig. 2. Online performance against model size, averaged over the datasets. The model size is relative to the stream length, whereas the online performance is measured relative to the top-performing method on each dataset without restriction on model size.

other ones. Moreover, it attains 90% of the best full-sampling methods while using only 1.5% of the data to represent the model. Because of the better performance exhibited by our methods with respect to the baselines at the same model size values, we can infer that our methods have a better way of choosing the data points that define their models.

Constant model size. In this section we propose a simple method for making the memory footprint bounded, even in the presence of an arbitrarily long data stream. When the model size reaches a given limit, the algorithm starts to discard the examples supporting the model that are judged to be less informative for the prediction task. More precisely, it is reasonable to discard the local classifiers that are making the largest number of mistakes. Removing local classifiers with a high mistake rate may help because: we are discarding classifiers that are making essentially random decisions; moreover, we make room for new classifiers that rely on fresh statistics (good in case of concept drift) and are possibly better positioned to capture a complex decision surface. In a nutshell, after the budget is reached, whenever a new ball is added an existing ball i is discarded according to the Laplace-corrected probability: $\mathbb{P}(i \,\text{discarded}) = \frac{m_i+1}{\sum_{j \in \mathcal{S}} m_j + |\mathcal{S}|}$; where m_i is the number of mistakes made by ball $i \in \mathcal{S}$.

We run the experiments in the same setting of the first set of experiments, where we did not make any restriction on the sub-sampling rate (`rate` $= 1$). We added to AUTO-ADJ a constant model size bound. In these experiments the budget parameters (as for `rate` in the previous experiments) is the maximum percentage of stream used as ball center. With respect to sub-sampling, here the algorithm has more control over the data points that support the model. Along the same lines of Figure 1, we show in Figure 2 the overall performance of the compared methods using the budget/rate values $\{1\%, 3\%, 5\%, 10\%\}$. AUTO-ADJ FIX clearly outperforms all the other methods. This is not surprising, as AUTO-ADJ FIX has a better way of choosing the data points supporting the model as opposed to the random selection imposed on the other methods.

VI. Conclusion and Future Works

We presented a new family of algorithms for nonparametric classification of data streams. Our more sophisticated algorithms feature the most appealing traits in stream mining applications: nonparametric classification, incremental learning, dynamic addition of new classes, small model size, fast prediction at testing time (logarithmic in the model size), essentially no parameters to tune. We empirically showed the effectiveness of our approach in different scenarios and against several standard baselines. In addition, we proved strong theoretical guarantees on the online performance of the most basic version of our approach. Future work will focus on investigating notions of local dimensions that allow to perform dimensionality reduction locally and incrementally.

Acknowledgements. RDR and NCB are supported by the MIUR through the ARS TechnoMedia Project under grant 2010N5K7EB 003.

References

[1] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[2] K. Clarkson. Nearest-neighbor searching and metric space dimensions. *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, 2005.

[3] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.*, 3:951–991, 2003.

[4] R. De Rosa and N. Cesa-Bianchi. Splitting with confidence in decision trees with application to stream mining. In *Int. Joint Conf. on Neural Networks*. IEEE, 2015.

[5] R. De Rosa, N. Cesa-Bianchi, I. Gori, and F. Cuzzolin. Online action recognition via nonparametric incremental learning. In *25th British Machine Vision Conf.*, 2014.

[6] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. of the 6th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.

[7] P. Duda, M. Jaworski, L. Pietruczuk, and L. Rutkowski. A novel application of hoeffding's inequality to decision trees construction for data streams. In *Int. Joint Conf. on Neural Networks*. IEEE, 2014.

[8] M. Feder, N. Merhav, and M. Gutman. Universal prediction of individual sequences. *IEEE Trans. Inform. Theory*, 38(4):1258–1270, 1992.

[9] Y. Freund and R. E. Schapire. Large margin classification using the Perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[10] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. A survey of classification methods in data streams. In *Data Streams*, pages 39–59. Springer, 2007.

[11] J. Gama and P. Kosina. Learning decision rules from data streams. In *Int. Joint Conf. on Artificial Intelligence*, pages 1255–1260, 2011.

[12] J. Gama, R. Sebastiao, and P. P. Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

[13] E. Hazan and N. Megiddo. Online learning with prior knowledge. In *Annual Conf. on Learning Theory*, pages 499–513. Springer, 2007.

[14] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. of KDD*, pages 97–106. ACM, 2001.

[15] S. Kpotufe and F. Orabona. Regression-tree tuning in a streaming setting. In *Advances in Neural Information Processing Systems 26*, pages 1788–1796, 2013.

[16] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proc. of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 798–807, 2004.

[17] F. Orabona. *DOGMA: a MATLAB toolbox for Online Learning*, 2009. Software available at http://dogma.sourceforge.net.

[18] J. Read, A. Bifet, G. Holmes, and B. Pfahringer. Scalable and efficient multi-label classification for evolving data streams. *Machine Learning*, 88(1-2):243–272, 2012.

[19] A. Shaker and E. Hüllermeier. Iblstreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3(4):235–249, 2012.

[20] I. Steinwart. On the influence of the kernel on the consistency of support vector machines. *J. Mach. Learn. Res.*, 2:67–93, 2002.

[21] Tsymbal. *Data sets with concept drift*, 2006. Available online at http://www.win.tue.nl/~mpechen/data/DriftSets/.

[22] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Dynamic integration of classifiers for handling concept drift. *Inf. Fusion*, 9(1):56–68, Jan. 2008.

[23] D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.

[24] P. Zhang, B. J. Gao, X. Zhu, and L. Guo. Enabling fast lazy learning for data streams. In *Proc. of ICDM*, pages 932–941. IEEE, 2011.

[25] X. Zhu. *Stream data mining repository*, 2010. Available online at http://www.cse.fau.edu/~xqzhu/stream.html.

APPENDIX A
PROOFS

We use the following well-known fact: if $p_t = \mathbb{P}(\widehat{y}_t = 1)$ for predicting $y_t \in \{0,1\}$ using a randomized label $\widehat{y}_t \in \{0,1\}$, then $\mathbb{P}(\widehat{y}_t \neq y_t) = |y_t - p_t|$. In the following, we say that a phase ends each time condition in line 15 of BASE is verified and use T_i to denote the time steps included in phase i. Finally, $\mathcal{S}_i$ denotes the maximum number of balls used in phase i. We use the following lemma from the ITBR analysis [15].

Lemma 1 ([15]): Suppose BASE is run with parameter $\widehat{C} \geq C_S$. Then for all phases $i \geq 1$ we have $i \leq d_i \leq d$ and for any $t \in T_i$ we have $|\mathcal{S}_i| \leq \widehat{C}\, 4^{d_i} \epsilon_t^{-d_i}$.

Define $\ell_t(p_t) = |p_t - y_t|$. Unlike the analysis in [15], here we cannot use a bias-variance decomposition. So, the key in the proof is to decompose the regret in two terms with behaviour similar to the bias and variance terms in the stochastic setting.

Lemma 2: Let d be the metric dimension of the set S of data points in the stream. Assume that $\widehat{C} \geq C_S$. Then, in any phase i and for any $f \in \mathcal{F}_L$ we have that

$$\sum_{t \in T_i} \Big(\ell_t(p_t) - \ell_t\big(f(\boldsymbol{x}_t)\big)\Big) \leq \Big(2\sqrt{\widehat{C}}\, 2^{d_i+1} + 1.5L\Big) n_i^{\frac{1+d_i}{2+d_i}}.$$

Proof: We use the notation $\boldsymbol{x}_t \to \boldsymbol{x}_s$ to say that $\boldsymbol{x}_t$ is assigned to a ball with center $\boldsymbol{x}_s$. We also denote by $n(\boldsymbol{x}_s)$ the number of points assigned to a ball of center $\boldsymbol{x}_s$. Define $p_s^* = \operatorname{argmin}_{p \in [0,1]} \sum_{t\,:\,\boldsymbol{x}_t \to \boldsymbol{x}_s} \ell_t(p)$. For each $\boldsymbol{x}_s$ in $\mathcal{S}_i$, we proceed by upper bounding the error as a sum of two components

$$\sum_{t\,:\,\boldsymbol{x}_t \to \boldsymbol{x}_s} \Big(\ell_t(p_t) - \ell_t\big(f(\boldsymbol{x}_t)\big)\Big) = \sum_{t\,:\,\boldsymbol{x}_t \to \boldsymbol{x}_s} \big(\ell_t(p_t) - \ell_t(p_s^*)\big) + \sum_{t\,:\,\boldsymbol{x}_t \to \boldsymbol{x}_s} \Big(\ell_t(p_s^*) - \ell_t\big(f(\boldsymbol{x}_t)\big)\Big).$$

Using the definition of p_s^* and the Lipschitzness of f, we have

$$\begin{aligned}\ell_t(p_s^*) - \ell_t(f(\boldsymbol{x}_t)) &\leq \ell_t(f(\boldsymbol{x}_s)) - \ell_t(f(\boldsymbol{x}_t)) \\ &\leq |f(\boldsymbol{x}_s) - f(\boldsymbol{x}_t)| \leq L\,\rho\,(\boldsymbol{x}_s, \boldsymbol{x}_t) \leq L\,\epsilon_t.\end{aligned}$$

The prediction strategy in each ball is equivalent to the approach followed in [8] (see also Exercise 8.8 in [1]). The only important thing to note is that the first prediction of the algorithm in a ball is made using the probability of the closest ball, even if it is further than ϵ_t, instead of at random as in the original strategy in [8]. It is easy to see that this adds an additional 0.5 to the regret stated in [8]. So we have

$$\sum_{t\,:\,\boldsymbol{x}_t \to \boldsymbol{x}_s} (\ell_t(p_t) - \ell_t(p_s^*)) \leq \sqrt{n(\boldsymbol{x}_s) + 1} + 1 \leq 2.5\sqrt{n(\boldsymbol{x}_s)}.$$

Hence overall we have

$$\sum_{t\,:\,\boldsymbol{x}_t \to \boldsymbol{x}_s} \Big(\ell_t(p_t) - \ell_t\big(f(\boldsymbol{x}_t)\big)\Big) \leq 2.5\sqrt{n(\boldsymbol{x}_s)} + L \sum_{t\,:\,\boldsymbol{x}_t \to \boldsymbol{x}_s} \epsilon_t.$$

Summing over all the $\boldsymbol{x}_s \in \mathcal{S}_i$, we have

$$\sum_{t \in T_i} \Big(\ell_t(p_t) - \ell_t\big(f(\boldsymbol{x}_t)\big)\Big) \leq 2.5 \sum_{s=1}^{|\mathcal{S}_i|} \sqrt{n(\boldsymbol{x}_s)} + L \sum_{t \in T_i} \epsilon_t. \quad (1)$$

For the first term in the r.h.s. of (1) we have

$$\sum_{s=1}^{|\mathcal{S}_i|} \sqrt{n(\boldsymbol{x}_s)} \leq |\mathcal{S}_i| \sqrt{\frac{1}{|\mathcal{S}_i|} \sum_{s=1}^{|\mathcal{S}_i|} n(\boldsymbol{x}_s)} \leq 2.5\sqrt{|\mathcal{S}_i| n_i}.$$

To bound $|\mathcal{S}_i|$ we use Lemma 1. While to bound the second term in (1) we have

$$\sum_{t \in T_i} \epsilon_t = \sum_{t=1}^{n_i} t^{-\frac{1}{2+d_i}} \leq \int_0^{n_i} \tau^{-\frac{1}{2+d_i}}\, d\tau \leq 1.5 n_i^{\frac{d_i+1}{2+d_i}}$$

where $n_i = |T_i|$. Putting all together and using again Lemma 1 we get the stated bound. ∎

Proof: (of Theorem 1) Let I denote the number of phases up to time T. Let $B \triangleq 2.5\sqrt{\widehat{C}}\, 2^d + 1.5L$. We use Lemma 2 in each phase and sum over the phases, to have

$$\begin{aligned}\sum_{t=1}^{T} (\ell_t(p_t) - \ell_t(p_s^*)) &= \sum_{i=1}^{I} \sum_{t \in T_i} (\ell_t(p_t) - \ell_t(p_s^*)) \\ &\leq B \sum_{i=1}^{I} n_i^{\frac{1+d}{2+d}} = B\, I \sum_{i=1}^{I} \frac{1}{I} n_i^{\frac{1+d}{2+d}} \leq B\, I \left(\sum_{i=1}^{I} \frac{n_i}{I}\right)^{\frac{1+d}{2+d}} \\ &= B\, I \left(\frac{T}{I}\right)^{\frac{1+d}{2+d}} \leq B d^{\frac{1}{2+d}} T^{\frac{1+d}{2+d}} \leq 1.26 B\, T^{\frac{1+d}{2+d}}\end{aligned}$$

where in the second inequality we use Jensen's inequality, and in the third one the first statement of Lemma 1. ∎